

“Todos os problemas na Ciência da Computação podem ser resolvidos adicionando um nível extra de indireção, exceto o problema de se ter muitos níveis de indireção” (David J. Wheeler).

Hierarquia de Memórias

Paulo Lisboa de Almeida



Precisamos de Memória

Memória para armazenar as instruções de programas e os dados.

Desejamos uma quantidade infinita de memória.

Mas não ter uma memória muito grande. Problemas:

- Físicos.
- Orçamentários.

Precisamos de Memória

Além de grande, desejamos uma memória rápida.

Pelo menos tão rápida quanto a CPU consegue solicitar informações.

Precisamos de Memória

Além de grande, desejamos uma memória rápida.

Pelo menos tão rápida quanto a CPU consegue solicitar informações.

Mas se não conseguimos pagar nem por uma memória grande, uma memória grande e rápida se torna impraticável.

O Quão Rápida a Memória Precisa Ser?

Segundo Hennessy e Patterson (2017).

Um processador Intel i7 7600 - 4 núcleos @4,1GHz em seu pico de operação:

- Faz duas referências de 64 bits a memória por core a cada ciclo de clock.

- Pode também demandar até $12,8 * 10^9$ instruções de 128 bits por segundo.

O Quão Rápida a Memória Precisa Ser?

Segundo Hennessy e Patterson (2017).

Um processador Intel i7 7600 - 4 núcleos @4,1GHz em seu pico de operação:

Faz duas referências de 64 bits a memória por core a cada ciclo de clock.

Pode também demandar até $12,8 * 10^9$ instruções de 128 bits por segundo.

A taxa de transferência entre a memória principal e o processador deveria ser de **409,6 GB/s** para suprir o processador.

O Quão Rápida a Memória Precisa Ser?

Segundo Hennessy e Patterson (2017).

Um processador Intel i7 7600 - 4 núcleos @4,1GHz em seu pico de operação:

Faz duas referências de 64 bits a memória por core a cada ciclo de clock.

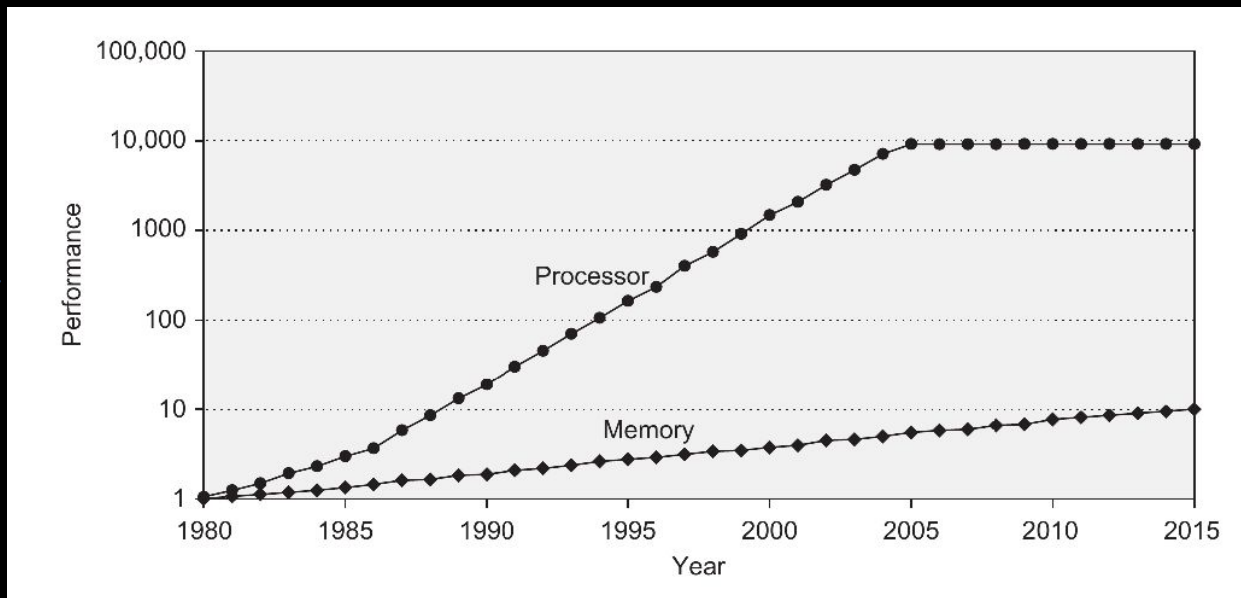
Pode também demandar até $12,8 * 10^9$ instruções de 128 bits por segundo.

A taxa de transferência entre a memória principal e o processador deveria ser de **409,6 GB/s** para suprir o processador.

As memórias DRAM comumente utilizadas em conjunto com esse processador, configuradas em *dual chanel* conseguem suprir apenas **34,1GB/s**.

Diferença de desempenho

Escala logarítmica.

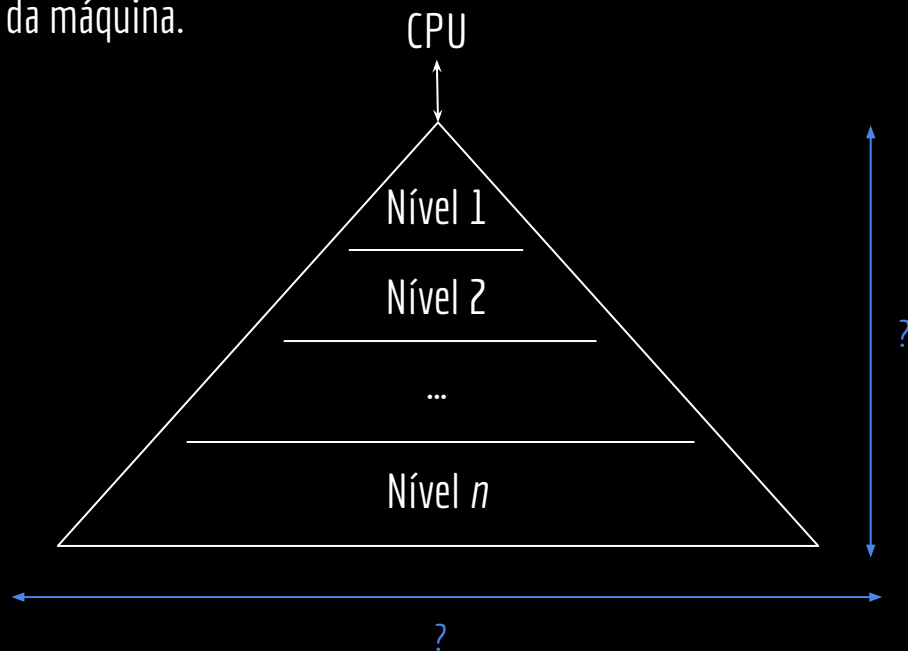


Diferença entre a velocidade em que o processador consegue processar instruções versus a velocidade em que a memória consegue suprir essas instruções. (Hennessy, Patterson; 2019).

Hierarquia de Memórias

Para amenizar esses problemas, é usada uma hierarquia de memórias, com n níveis.

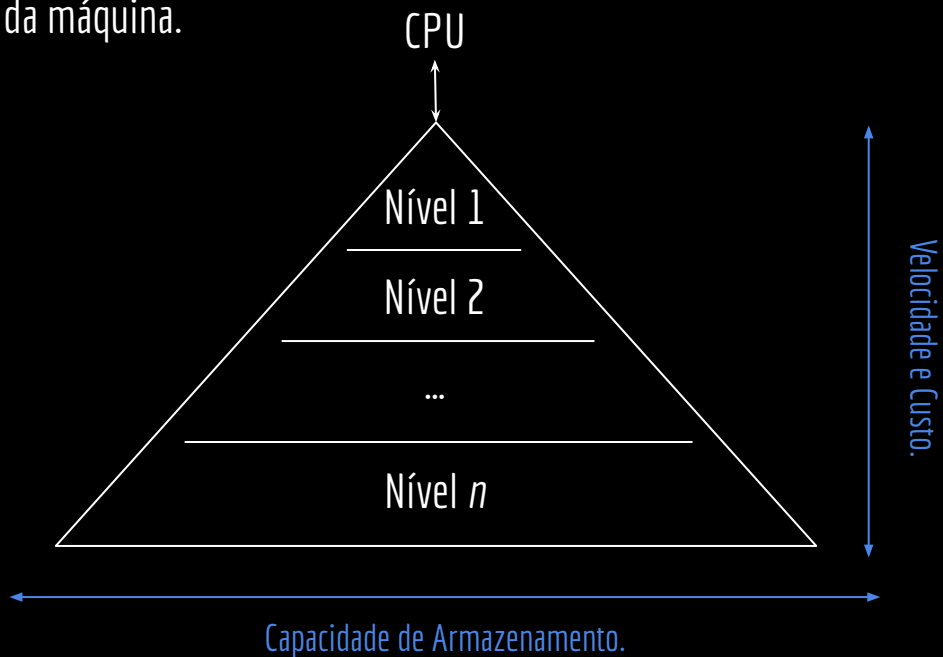
A quantidade de níveis vai depender da arquitetura da máquina.



Hierarquia de Memórias

Para amenizar esses problemas, é usada uma hierarquia de memórias, com n níveis.

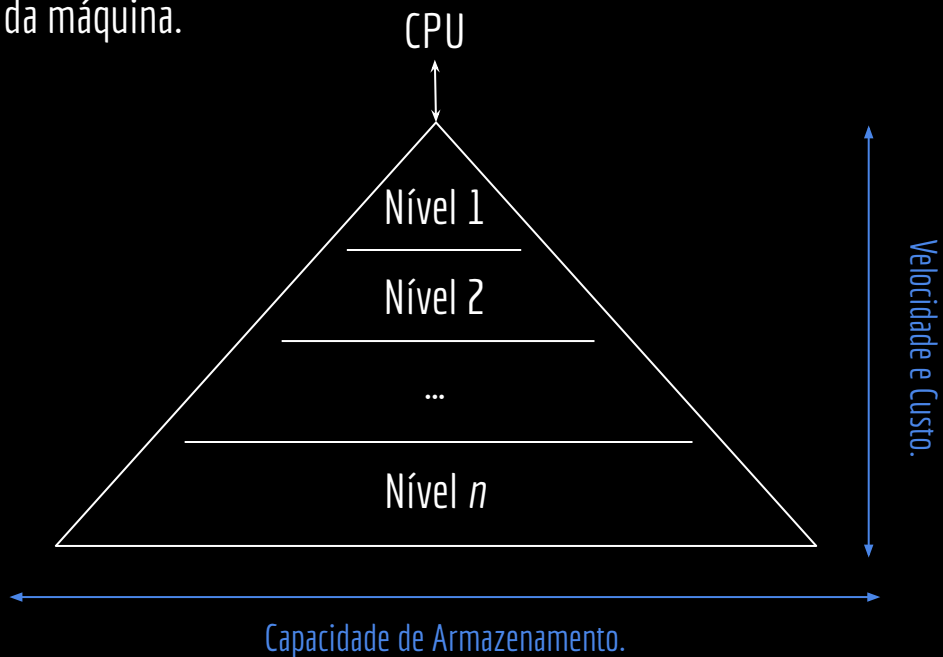
A quantidade de níveis vai depender da arquitetura da máquina.



Hierarquia de Memórias

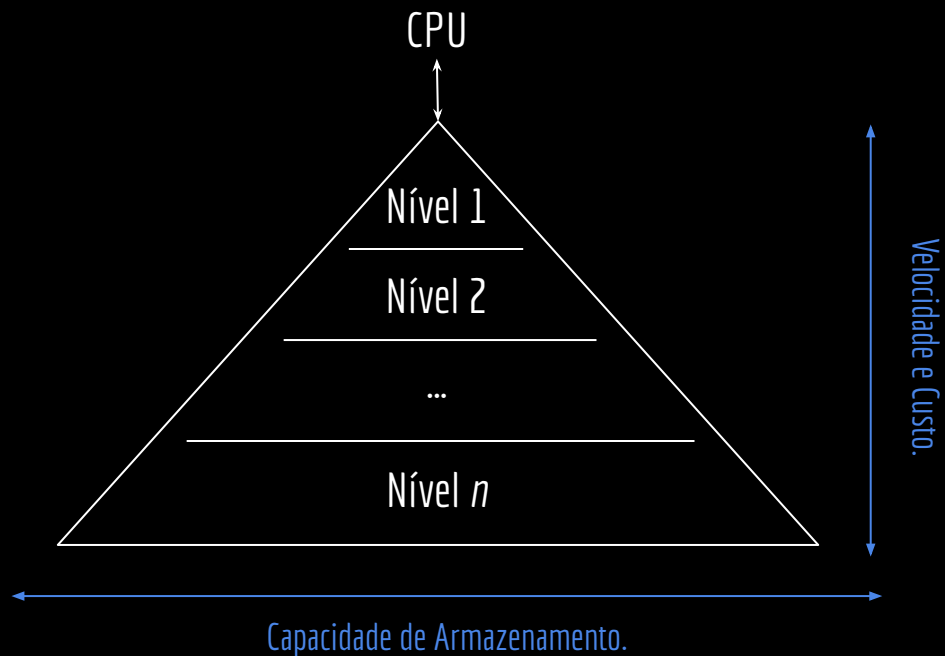
Para amenizar esses problemas, é usada uma hierarquia de memórias, com n níveis.

A quantidade de níveis vai depender da arquitetura da máquina.



Hierarquia de Memórias

Tomando como base o seu computador x86-64, você consegue especificar quais memórias estão em quais níveis?



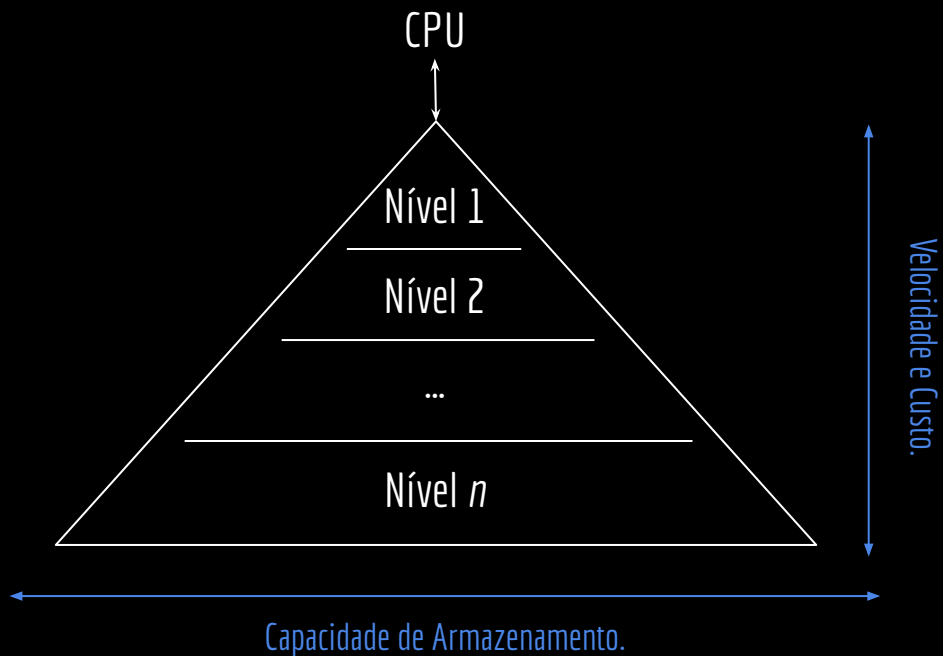
Hierarquia de Memórias

Tomando como base o seu computador x86-64, você consegue especificar quais memórias estão em quais níveis?

Caches.

Memória Principal (DRAM).

HD ou SSD (ou ambos).



Acesso aos níveis

Na maioria das implementações, temos uma **hierarquia real**.

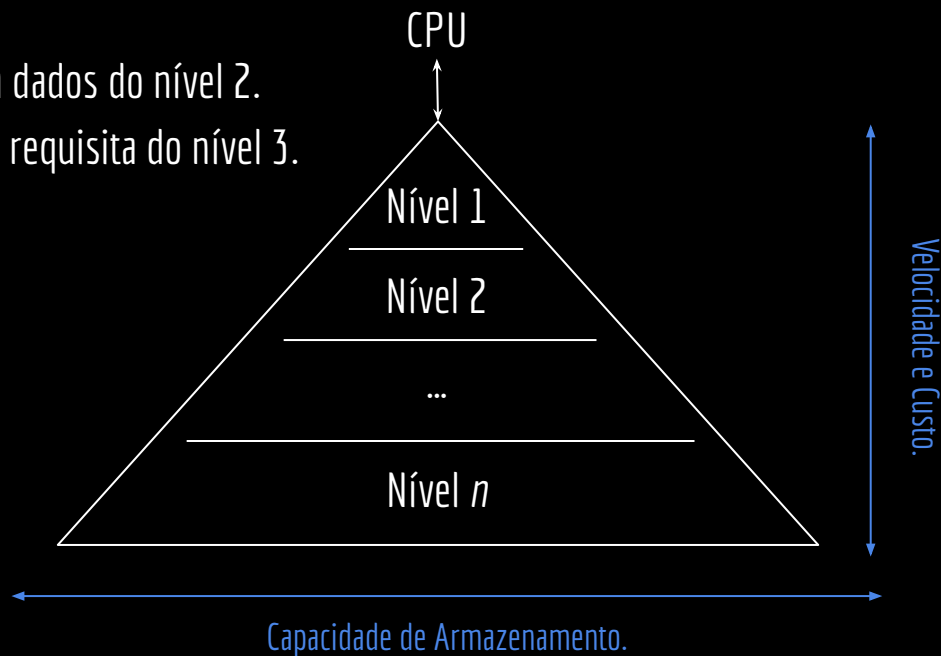
A CPU acessa os dados apenas no nível 1.

O nível 1 serve dados para a CPU, e requisita dados do nível 2.

0 nível 2 serve dados para o nível 1, e requisita do nível 3.

...

Não pulamos níveis em uma hierarquia real.



Uma questão de custos

Tecnologia	Tempo de acesso típico	Dólares por GB em 2020
SRAM	0.5 - 2.5 ns	\$500 - \$1000
DRAM	50 - 70 ns	\$3 - \$6
Flash	5.000 - 50.000 ns	\$0,03 - \$0,12
Disco Magnético	5.000.000 - 20.000.000 ns	\$0,01 - \$0,02

Patterson, Hennessy. Computer Organization and Design RISC-V Edition. 2020 (Ed. Americana).

Uma questão de custos

Tecnologia	Tempo de acesso típico	Dólares por GB em 2020
SRAM	0.5 - 2.5 ns	\$500 - \$1000
DRAM	50 - 70 ns	\$3 - \$6
Flash	5.000 - 50.000 ns	\$0,03 - \$0,12
Disco Magnético	5.000.000 - 20.000.000 ns	\$0,01 - \$0,02

Patterson, Hennessy. Computer Organization and Design RISC-V Edition. 2020 (Ed. Americana).

Tecnologia	Tempo de acesso típico	Dólares por GiB em 2012
SRAM	0.5 - 2.5 ns	\$500 - \$1000
DRAM	50 - 70 ns	\$10 - \$20
Flash	5.000 - 50.000 ns	\$0,75 - \$1,00
Disco Magnético	5.000.000 - 20.000.000 ns	\$0,05 - \$0,1

Patterson, Hennessy. Computer Organization and Design. 5a ed, 2014 (Ed. Americana).

Tecnologias envolvidas

Considerando a hierarquia comumente encontrada nos x86-64.

Caches: SRAM (static random access memory).

Construídas utilizando transistores com retroalimentação (de forma similar aos flip-flops).

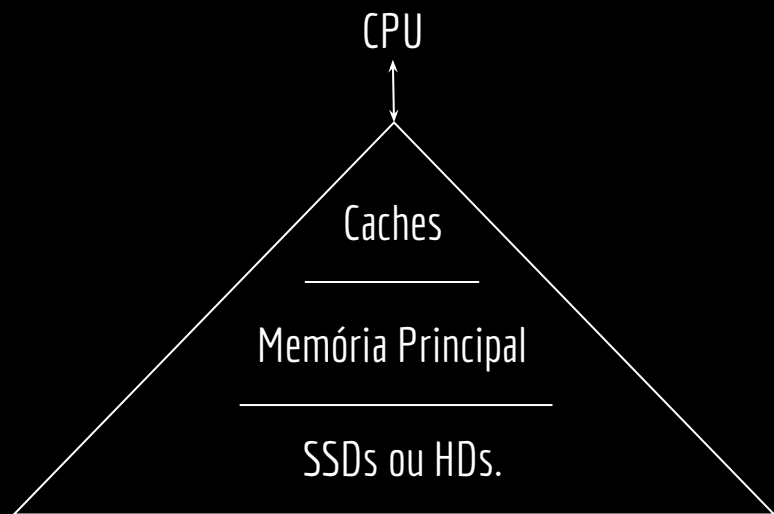
Memória Principal: DRAM (dynamic random access memory).

Memórias capacitivas.

Capacitores carregados/descarregados indicam o estado do bit.

SSD ou HD.

Flashes NAND ou NOR, ou discos magnéticos.



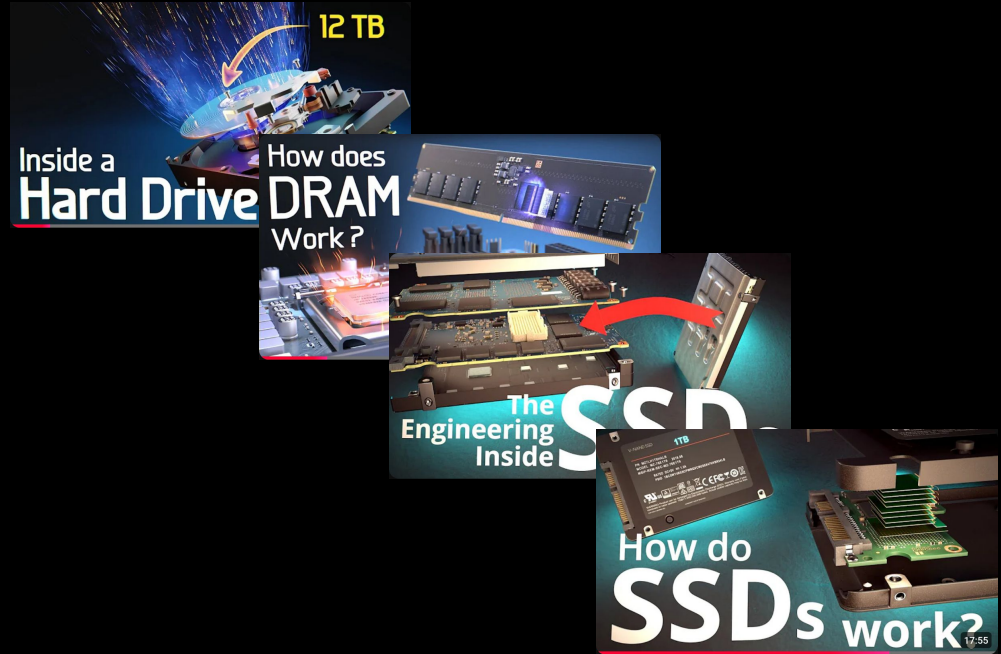
Assista

www.youtube.com/watch?v=wtdnatmVdlg&t=4s

www.youtube.com/watch?v=7J7X7aZvMXQ&t=382s

www.youtube.com/watch?v=r-SivgEpA1Q

www.youtube.com/watch?v=5Mh3o886qpg&t=789s

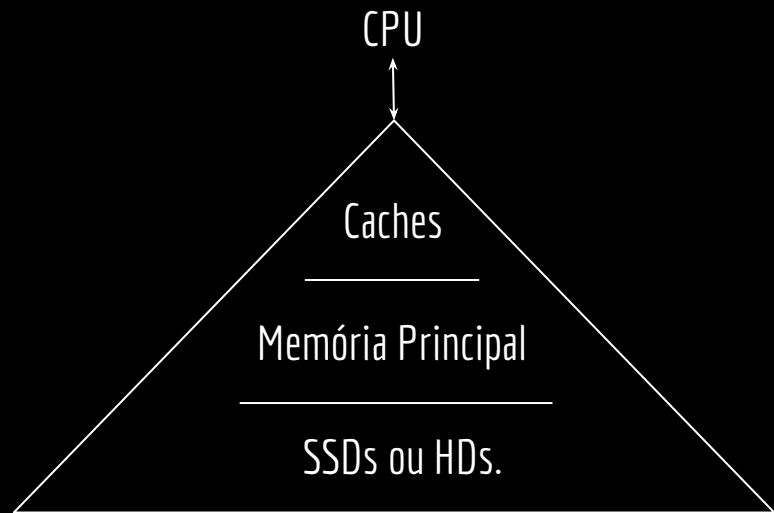


A Memória Principal

A memória principal fica entre os níveis da cache e armazenamento permanente.

Cache: foco em **velocidade**.

Armazenamento permanente: foco em **espaço**.



Armazenamento permanente

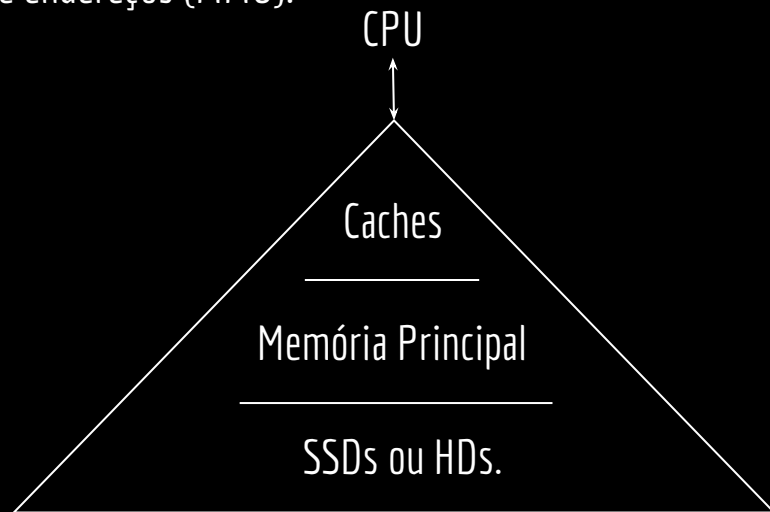
Utilizar o armazenamento permanente como se fosse a memória principal implica na utilização do conceito de **paginação**.

Tema para a disciplina de Sistemas Operacionais.

Envolve muito do hardware.

Exemplo: uso de interrupções e de unidades de tradução de endereços (MMU).

Veremos o básico sobre paginação no futuro.



Memória Cache

Começando com uma memória cache básica.

Enquanto os computadores atuais têm comumente 32GB de memória principal (DRAM), eles dificilmente possuem mais de 100MB de cache (SRAM).

Essentials	
Product Collection	Intel® Core™ i7 Processors (14th gen)
Code Name	Products formerly Raptor Lake
Vertical Segment	Desktop
Processor Number ⓘ	i7-14700
Lithography ⓘ	Intel 7
Recommended Customer Price ⓘ	\$384.00-\$394.00
CPU Specifications	
Total Cores ⓘ	20
# of Performance-cores	8
# of Efficient-cores	12
Total Threads ⓘ	28
Max Turbo Frequency ⓘ	5.4 GHz
Intel® Turbo Boost Max Technology 3.0 Frequency ¹ ⓘ	5.4 GHz
Performance-core Max Turbo Frequency ⓘ	5.3 GHz
Efficient-core Max Turbo Frequency ⓘ	4.2 GHz
Performance-core Base Frequency ⓘ	2.1 GHz
Efficient-core Base Frequency ⓘ	1.5 GHz
Cache ⓘ	33 MB Intel® Smart Cache
Total L2 Cache	28 MB

Memória Cache

Começando com uma memória cache básica.

Enquanto os computadores atuais têm comumente 32GB de memória principal (DRAM), eles dificilmente possuem mais de 100MB de cache (SRAM).

É comum encontrarmos processadores para servidores com quase 100MB ou mais de cache, mas dificilmente chegamos em 200MB.

A diferença entre as capacidades ainda é gigantesca.

Essentials	
Product Collection	Intel® Xeon® CPU Max Series
Code Name	Products formerly Sapphire Rapids HBM
Vertical Segment	Server
Processor Number ⓘ	9468
Lithography ⓘ	Intel 7
Recommended Customer Price ⓘ	\$9900.00
CPU Specifications	
Total Cores ⓘ	48
Total Threads ⓘ	96
Max Turbo Frequency ⓘ	3.50 GHz
Processor Base Frequency ⓘ	2.10 GHz
Cache ⓘ	105 MB
Intel® UPI Speed	16 GT/s
Max # of UPI Links ⓘ	4
TDP ⓘ	350 W

Transparência

Desejamos transparência.

O programador sempre vai assumir que o programa está na memória principal, e não vai programar explicitamente para os demais níveis.

Se o programa realmente está na memória principal, ou está na cache, ou no disco é problema do:

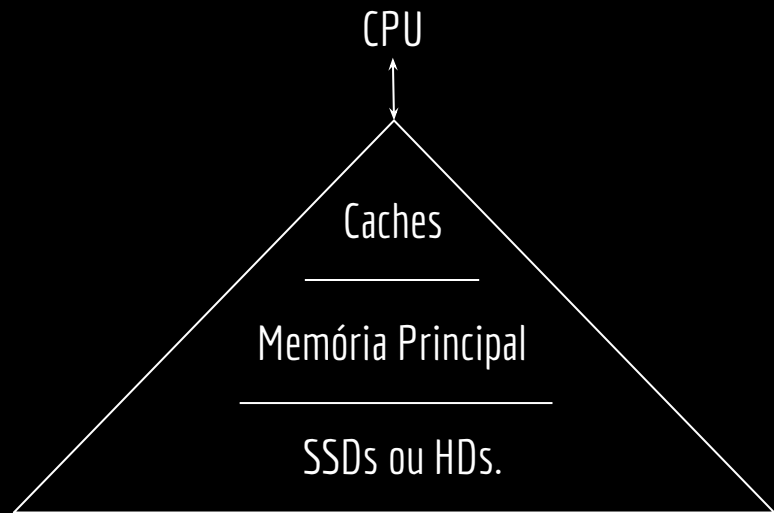
- Hardware no caso da memória cache.
- Hardware + Software (S.O.) no caso do Disco/SSD (paginação).

Memória Cache

Como podemos mapear a memória principal para dentro da cache de maneira automática?

Que regras seguir? Qual dado é importante?

Ideias?



Localidade temporal e espacial

Se analisarmos vários programas, veremos que temos dois tipos de localidade:

- Localidade temporal.
- Localidade espacial.

Localidade Temporal

Se um item da memória é acessado, é muito provável que ele seja acessado novamente num futuro próximo.

Exemplo:

Quando armazenamos um item na pilha para fazer uma chamada de função, vamos ler ele novamente quando a função terminar.

Localidade Espacial

Se acessamos um item da memória, é muito provável que seus vizinhos também sejam acessados.

Exemplos:

Nossos programas são (quase) sequenciais.

Quando a instrução i é carregada, é provável que $i+1$, $i+2$, ... também sejam úteis.

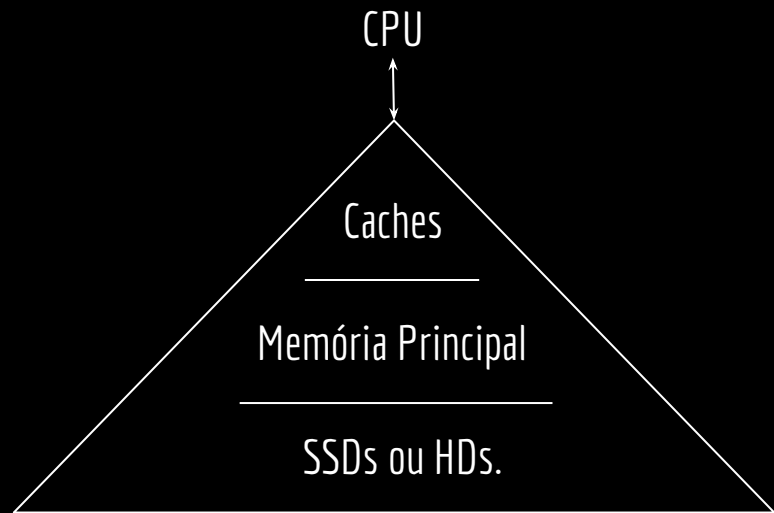
Quando operamos no elemento k de um vetor, o elementos $k+1$, $k+2$, ... provavelmente também serão úteis.

Memória Cache

A memória cache é muito menor que a memória principal.

Necessário tirar vantagem das localidades espacial e temporal.

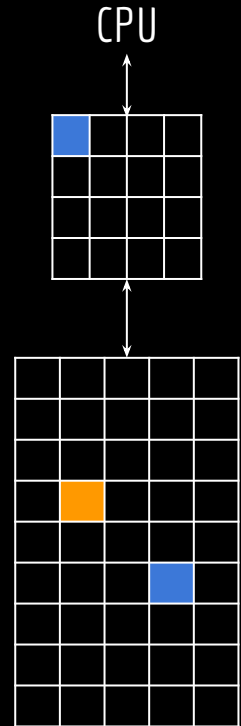
Manter na cache os **dados utilizados mais recentemente, e seus vizinhos**.



Mapeamento

Dividir as memórias (principal e cache) em blocos de **mesmo tamanho**.

Obviamente, a memória cache suporta menos blocos que a principal.



Mapeamento

Quando o processador requisita um dado e:

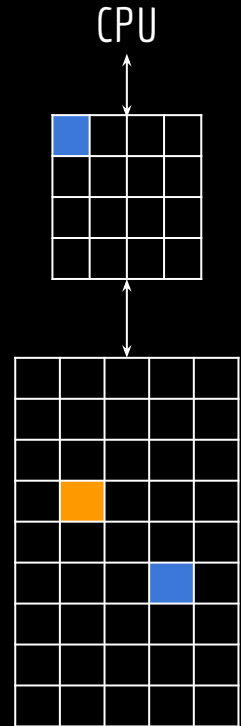
O dado está presente em um bloco no nível de memória mais alto (cache nesse caso).

Temos um **hit** (acerto).

Se o dado não está presente.

Temos um **miss** (erro).

Necessário tempo para carregar o bloco.



Medidas de Desempenho

hit rate (ou hit ratio).

Fração dos acessos a memória nos quais tivemos um hit.

$$hitRate = hits/totalAcessos$$

miss rate (ou miss ratio).

Fração dos acessos a memória nos quais tivemos um miss.

$$MissRate = 1 - hitRate$$

Medidas de Desempenho

Hit time

Tempo de hit.

Tempo necessário para acessar um bloco de memória no nível alto, considerando ainda o custo para verificar se temos ou não um hit.

Miss penalty

Penalidade de miss.

Tempo necessário para carregar/substituir um bloco do nível mais alto por um do nível logo abaixo.

Cache

Vamos considerar como cache a memória que está no nível mais alto.

No processador RISC-V, poderíamos simplesmente substituir as memórias de dados e instruções por caches.

No entanto, em um âmbito mais geral, cache também pode se referir a qualquer sistema de armazenamento que se beneficia da localidade de acesso.

Ex.: Muitos HDs possuem uma pequena cache interna, onde os dados mais requisitados são salvos.

Cache diretamente mapeada

Considere que:

O processador sempre requisita **uma palavra**.

Cada bloco da cache armazena exatamente **uma palavra**.

Quando o processador requisita uma palavra X_n que **não** está na cache.

Ocorreu um miss.

A palavra então é carregada para a cache.

Cache

X_4
X_1
X_{n-2}
X_{n-7}
...
X_3

Antes do Miss.

X_4
X_1
X_{n-2}
X_{n-7}
...
X_n
X_3

Depois do Miss.

Cache diretamente mapeada

Em uma cache **diretamente mapeada**.

Cada endereço de memória é mapeado para exatamente uma posição na cache.

A cache deve suportar uma potência de 2 de palavras.

O mapeamento é trivial.

Para uma cache de 2^n palavras, utilizamos os n bits mais baixos dos endereços para mapear a cache.

Exemplo

Cache de $2^3=8$ entradas.

Utilizamos os 3 bits menos significativos dos endereços de memória para endereçar a cache.

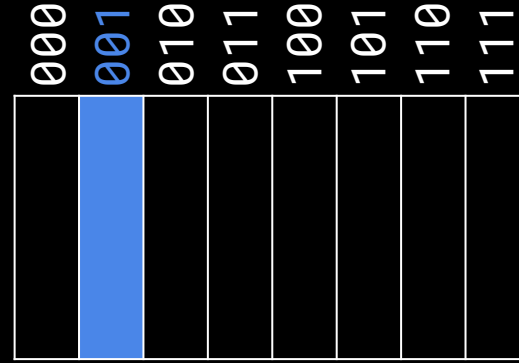
Cache

	000
	001
	010
	011
	100
	101
	110
	111

Memória Principal

00000	
00001	
00010	
00011	
00100	
00101	
00110	
00111	
01000	
01001	
01010	
01011	
01100	
01101	
01110	
01111	
11000	
11001	
11010	
11011	

Exemplo Cache 8 entradas



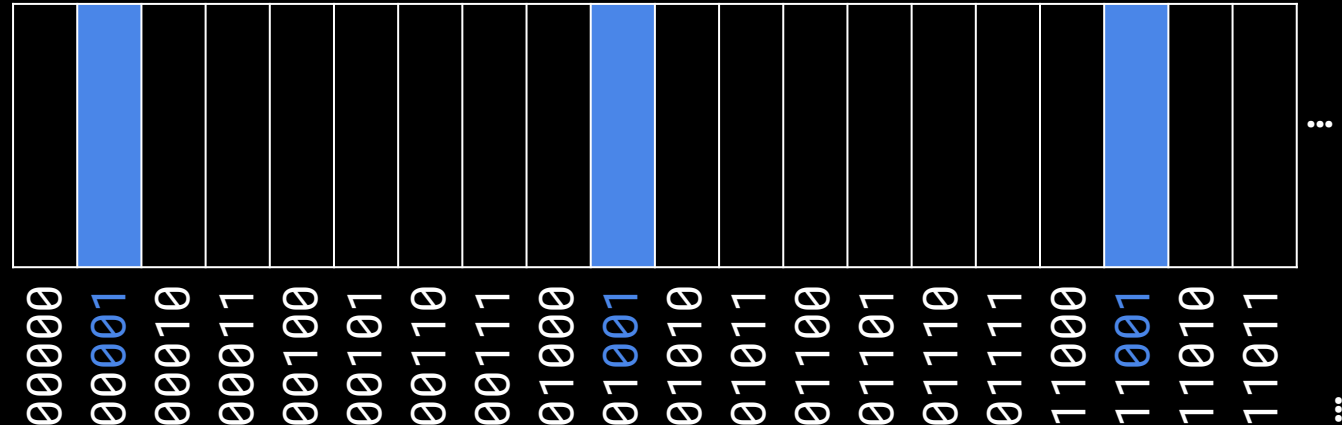
Se o processador solicitar o endereço 00001_2 .

O processador busca o dado no endereço 001_2 da cache.

A palavra pode ou não estar neste endereço.

Qual palavra da memória está nesse endereço?

- 00001_2
- 01001_2
- 10001_2
- 11001_2
- ...



Soluções?

Tag

Além da palavra, cada endereço da cache deve conter o **tag** da palavra.

Bits mais altos da palavra, que não foram utilizados para mapear a cache.

Exemplo

Palavras de 32 bits.

Cache de 8 palavras (3 bits para endereço).

Memória de 256 palavras (8 bits para endereço).

O dado no endereço $0010\ 1101_2$ está na cache.

Cache																			
	<table><tr><th>Tag</th><th>Palavra</th></tr><tr><td>000</td><td></td></tr><tr><td>001</td><td></td></tr><tr><td>010</td><td></td></tr><tr><td>011</td><td></td></tr><tr><td>100</td><td></td></tr><tr><td>101</td><td>Dado9</td></tr><tr><td>110</td><td></td></tr><tr><td>111</td><td></td></tr></table>	Tag	Palavra	000		001		010		011		100		101	Dado9	110		111	
Tag	Palavra																		
000																			
001																			
010																			
011																			
100																			
101	Dado9																		
110																			
111																			
	<table><tr><td>5 bits</td><td>32 bits</td></tr></table>	5 bits	32 bits																
5 bits	32 bits																		

Memória Principal

Palavra	
...	...
0010 0101	Dado1
0010 0110	Dado2
0010 0111	Dado3
0010 1000	Dado4
0010 1001	Dado5
0010 1010	Dado6
0010 1011	Dado7
0010 1100	Dado8
0010 1101	Dado9
0010 1110	Dado10
...	...
	32 bits

Exemplo

Agora sabemos que é o dado do endereço $0010\ 1101_2$, que está na cache, e não o do endereço $0010\ 0101_2$, por exemplo.

Cache																			
	<table><tr><th>Tag</th><th>Palavra</th></tr><tr><td>000</td><td></td></tr><tr><td>001</td><td></td></tr><tr><td>010</td><td></td></tr><tr><td>011</td><td></td></tr><tr><td>100</td><td></td></tr><tr><td>101</td><td>Dado9</td></tr><tr><td>110</td><td></td></tr><tr><td>111</td><td></td></tr></table>	Tag	Palavra	000		001		010		011		100		101	Dado9	110		111	
Tag	Palavra																		
000																			
001																			
010																			
011																			
100																			
101	Dado9																		
110																			
111																			

Memória Principal

		Palavra
...		...
0010	0101	Dado1
0010	0110	Dado2
0010	0111	Dado3
0010	1000	Dado4
0010	1001	Dado5
0010	1010	Dado6
0010	1011	Dado7
0010	1100	Dado8
0010	1101	Dado9
0010	1110	Dado10
...		...
		32 bits

Validade da cache

Acabamos de ligar a máquina.

A cache é inicializada com lixo.

Palavras e tags aleatórias.

Qual o problema? Como resolver?

Bit de validade

Bit de validade para cada entrada da cache.

Se o bit é 0, desconsiderar a cache.

Exemplo de cache de 8 palavras.

	val?	Tag	Palavra
000	1	00101	Dado X
001	0	lixo	lixo
010	0	lixo	lixo
011	1	11111	Dado Y
100	1	00000	Dado Z
101	1	00101	Dado W
110	0	lixo	lixo
111	1	10101	Dado K

Miss ou Hit?

Quando o processador precisa ler uma informação no endereço X .

Utiliza os bits mais baixos de X para encontrar a entrada na cache.

Os demais bits de X são comparados com o tag.

O bit de validade é comparado.

Se tudo isso bater, temos um **hit**.

Caso contrário, temos um **miss**.

	val?	Tag	Palavra
000	1	00101	Dado X
001	0	lixo	lixo
010	0	lixo	lixo
011	1	11111	Dado Y
100	1	00000	Dado Z
101	1	00101	Dado W
110	0	lixo	lixo
111	1	10101	Dado K

Miss

Em caso de miss.

A palavra é solicitada do nível de memória inferior.

Carregada para o endereço de memória correto da cache.

Caso já haja algo nesse endereço da cache, **o dado é substituído**.

O campo tag é atualizado.

O bit de validade é *setado*.

A palavra é enviada ao processador.

	val?	Tag	Palavra
000	1	00101	Dado X
001	0	lixo	lixo
010	0	lixo	lixo
011	1	11111	Dado Y
100	1	00000	Dado Z
101	1	00101	Dado W
110	0	lixo	lixo
111	1	10101	Dado K

Falácias

“Para programar você não precisa conhecer os detalhes da hierarquia de memória.”

Realmente você sempre pode considerar que o programa está na memória principal, e que ela é um vetor com endereços físicos fixos.

Tudo vai funcionar.

Mas para aplicações que **exigem o mínimo de desempenho**, saber como os diferentes níveis de memória operam é **fundamental**.

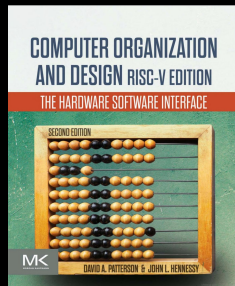
Exercícios

1. Considere uma cache de 8 entradas inicialmente vazia, e que os seguintes endereços são solicitados em ordem: 10110_2 , 11010_2 , 10110_2 , 11010_2 , 10000_2 , 00011_2 , 10000_2 , 10010_2 , 10000_2 .

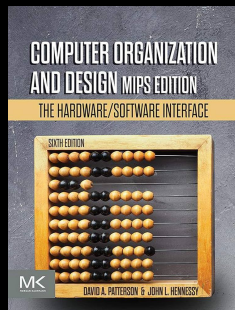
Qual o estado final da cache depois de solicitar todos os endereços? Para simplificar, considere que o dado no endereço X é DadoX.
2. Considere uma memória principal com endereços que ocupam 16 bits, e uma memória cache com 256 entradas. Considere que cada endereço de memória principal/cache suporta uma palavra de 32 bits. Qual é o tamanho da memória cache considerando apenas as palavras da memória que ela suporta? Qual é o tamanho total da memória cache? Qual a percentagem de dados na cache que formam os dados de controle (e.g. bit de validade) quando comparados ao percentual de dados de palavras da cache?

Referências

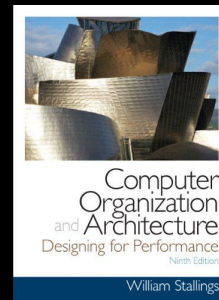
Patterson, Hennessy.
Computer Organization and
Design RISC-V Edition: The
Hardware Software
Interface. 2020.



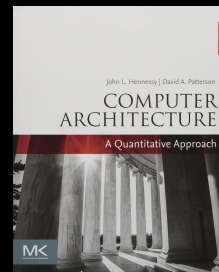
Patterson, Hennessy. Computer
Organization and Design MIPS
Edition: The Hardware/Software
Interface. 2020.



Stallings, W. Organização de
Arquitetura de Computadores.
10a Ed. 2016.



Hennessy, Patterson.
Arquitetura de Computadores:
uma abordagem quantitativa.
2019.



Licença

Esta obra está licenciada com uma Licença [Creative Commons Atribuição 4.0 Internacional](https://creativecommons.org/licenses/by/4.0/).